*Université A.Mira, Bejaia*
*Faculté des Sciences Exactes*
*Département d'Informatique*

# Cours de Compilation

## Introduction

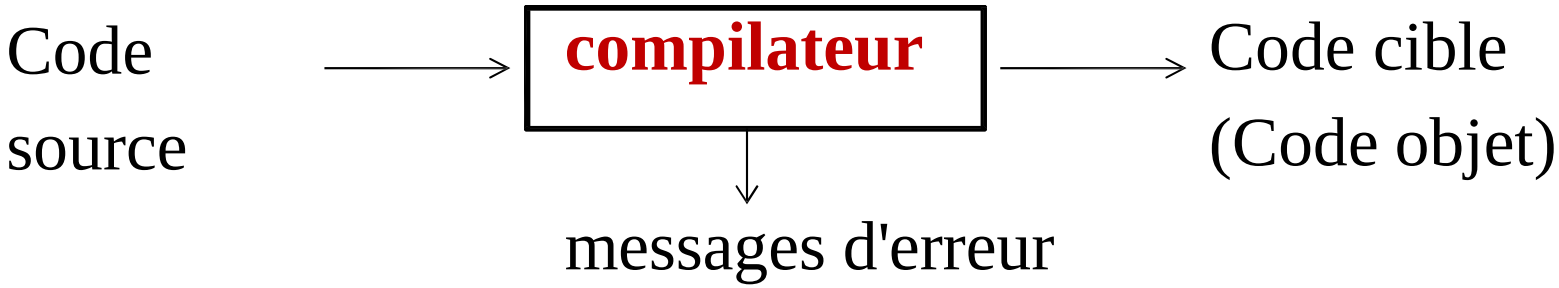
*Mme D.Boulahrouz*
*boukredera@hotmail.com*

# 1- Les Compilateurs

## 1.1- **Définition**:

*Un **compilateur** est un programme qui **lit** un programme écrit dans un premier langage (le langage source) et le **traduit** en un programme équivalent dans un autre langage (le langage cible).*

*Le compilateur doit aussi **vérifier** que le programme a un certain sens et signaler les erreurs qu'il détecte.*



# 1- Les Compilateurs

Il y' a deux parties (phases) dans la compilation: l'**analyse** et la **synthèse**.

**La partie analyse** partitionne le programme source en ses constituants et en crée une représentation intermédiaire
*Elle détecte et signale les erreurs (syntaxiques, typage, ...) et peut donc échouer.*

**La partie synthèse** construit le programme cible a partir de cette représentation intermédiaire.
*N'échoue pas.*

# 2- Les Phases d'Analyse

L'**analyse** est constituée de **3 parties**:

2.1. **L'analyse lexicale** (Scanner): le flot de caractères formant le programme source est **lu** de gauche a droite et **groupé** en ***lexèmes*** (mots), qui sont des suites de caractères ayant une signification collective.

# Exemple d'analyse

Soit l'instruction: A = B1 + B2 * 60

**L'analyse lexicale**:

1. L'identificateur "A"
2. Le symbole d'affectation "="
3. L'identificateur "B1"
4. Le signe "+"
5. L'identificateur "B2"
6. Le signe de multiplication "*"
7. Le nombre "60"

*Remarque*: les blancs qui séparent les mots sont éliminés.

# 2- Les Phases d'Analyse

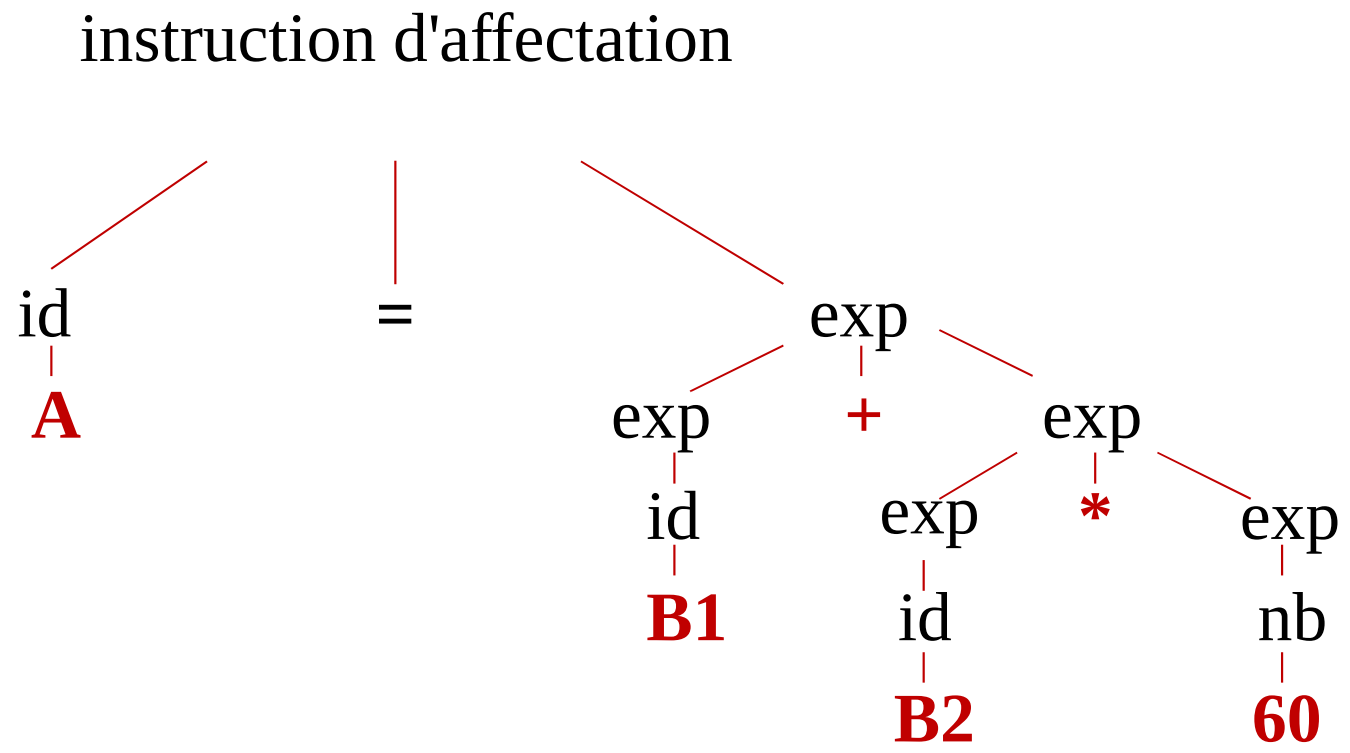
L'**analyse** est constituée de **3 parties**:

**2.2** **L'analyse syntaxique** (Parser): les unités lexicales sont regroupés hiérarchiquement dans des collections imbriquées (phrases) ayant une signification collective et conformément à la syntaxe du langage, décrite par une grammaire non contextuelle de type (2).

Cette phase permet de construire un arbre syntaxique qui représente la syntaxe du programme source.

# Exemple d'analyse (suite)

*L'arbre syntaxique correspondant à*: **A = B1 + B2 * 60**




Remarque: L'arbre abstrait est une forme compacte de l'arbre syntaxique

# 2- Les Phases d'Analyse

L'**analyse** est constituée de **3 parties**:

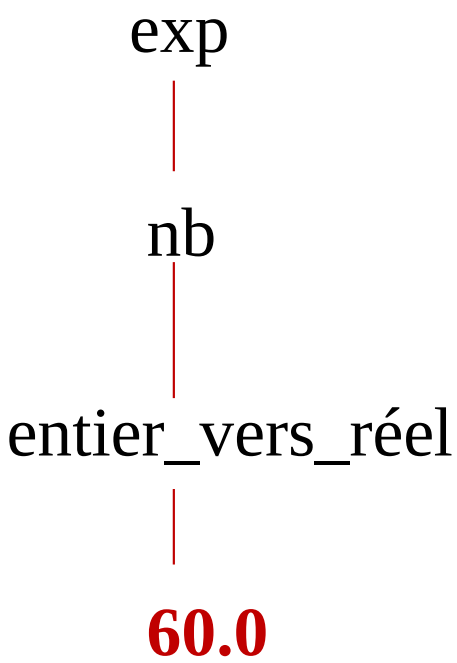
**2.3**. **L'analyse sémantique**: Son rôle consiste à

- **vérifier** que les composants du programme source sont agencés d'une façon significative ;
- **Recueillir** les informations sur les types des identificateurs et mettre à jour la table des symboles ;
- **Assurer** le contrôle des types ;
- **Vérifier** le nombre et les types des arguments lors des appels de procédures et de fonctions.

# Exemple d'analyse (suite)

**L'analyse sémantique:**

Si nous déclarons A, B1 et B2 des réels, alors nous devons convertir l'entier **60** en réel **60.0**



# 2- Les Phases d'Analyse

L'**analyse** est constituée de **3 parties**:

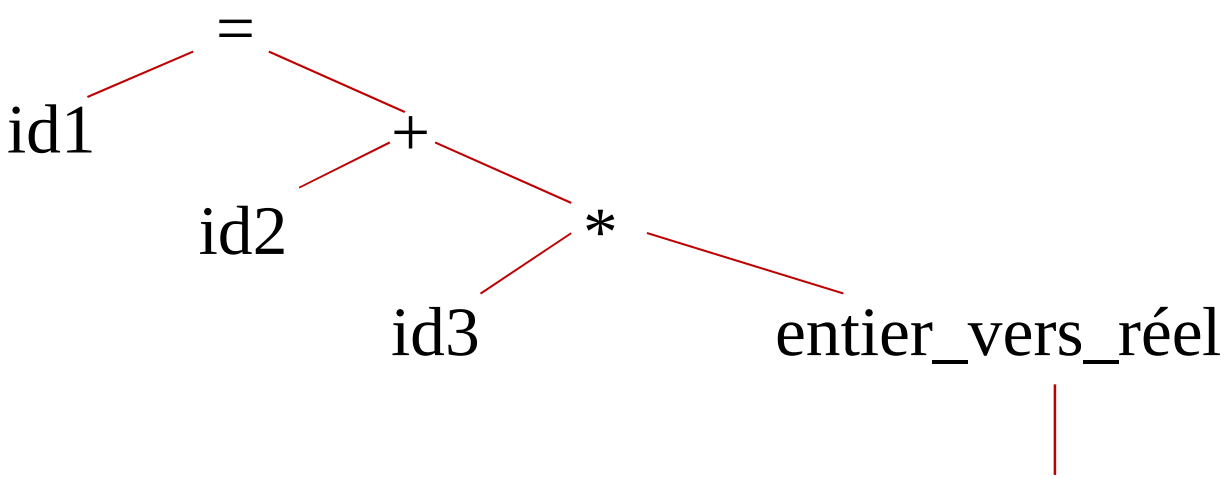
**2.4**. **Production du code intermédiaire**: Après les trois phases d'analyse, le compilateur génère une représentation ente le programme source et le programme cible. Cette représentation doit être facile à produire et facile à traduire en code cible.

Exemple :
La forme post-fixée :
I = 5 + i * 3 => i 5 i 3 * + =

# Exemple d'analyse (suite)

**Le code intermédiaire:**

L'arbre abstrait:




```
temp1 := entier_vers_réel(60);
temp2 := id3 * temp1;
temp3 := id2 + temp2;
id1   := temp3;
```

# 3- Les Phases de Synthèse

**3.1- Optimisation du code Intermédiaire**:

Amélioration du code intermédiaire pour que le code final s'exécute plus rapidement et utilise le minimum de mémoire.

```
temp1 := id3 * 60.0;
id1 := id2 + temp1;
```

# Exemple de Synthèse

**3.2- Génération du code**:

Production du code cible en langage d'assemblage.

L'aspect crucial est l'assignation des variables aux registres du processeur.

Si nous utilisons 2 registres *R1* et *R2*, la traduction pourrait être :

```
MOVF R2,id3
MULF R2,#60.0
MOVF R1,id2
ADDF R1,R2
MOVF id1,R1
```

F: nombre en virgule flottante

#: 60.0 doit être traité comme une constante

# 4- Regroupement des phases:

Souvent on regroupe les phases en **2 parties**:

**4.1- La partie frontale**: constituée des phases qui dépendent principalement du langage source:

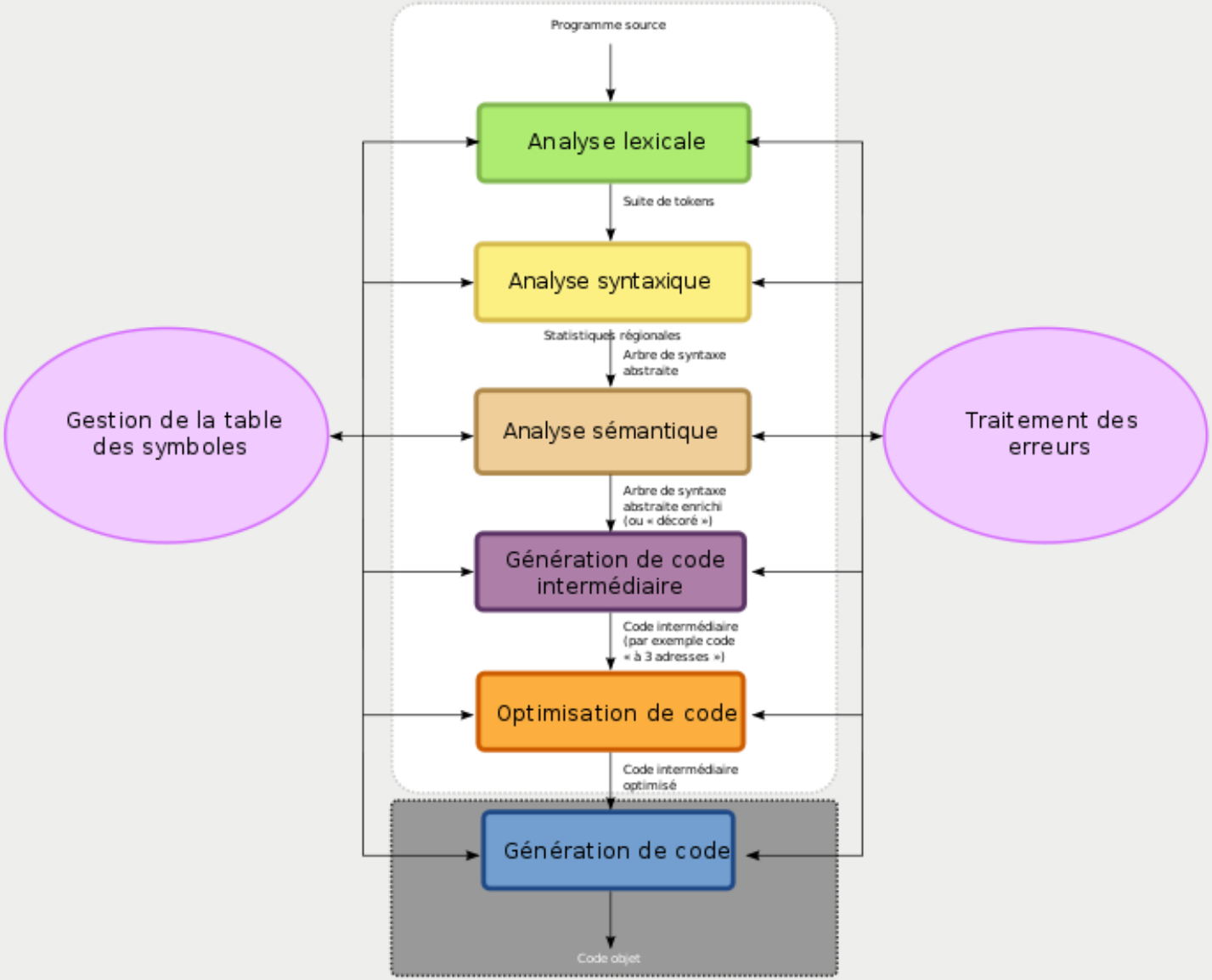
- l'analyse **lexicale**,
- l'analyse **syntaxique**,
- l'analyse **sémantique**
- la création de la **table des symboles**,
- la production du **code intermédiaire**
- elle inclut le traitement des erreurs associées a chacune de ces phases.

# 4- Regroupement des phases:

## 4.2- La partie finale:

Elle est constituée des phases qui dépendent de la machine cible.

- **Optimisation** du code intermédiaire,
- production **du code final**,
- gestion de la table des symboles,
- traitement des erreurs.




Chaîne de compilation

Phase frontale (front end)

Phase finale (back end)

# Les Compilateurs

**Gestion de la table des symboles**:

Une table des symboles est une structure de données contenant un enregistrement pour chaque identificateur, muni de champs pour ses attributs (*emplacement mémoire*, *son type*, *sa protée*..).

Pour les fonctions, la table des symboles garde le nombre et les types de ses arguments, le mode de passage de chacun d'eux et la valeur de retour.

# Les Compilateurs

**Gestion de la table des symboles**:

| indice | nom | adresse | ... | |
|---|---|---|---|---|
| 1 | A | | | |
| 2 | B1 | | | |
| 3 | B2 | | | |
| 4 | ... | | | |

# Les Compilateurs

**Traitement des erreurs**:

Chaque phase peut rencontrer des erreurs.

Après avoir détecté une erreur, une phase doit la traiter de telle façon que la compilation puisse continuer et que d'autres erreurs dans le programme puissent être détectées.

# 5- Les Interprètes

Au lieu de produire un programme cible, un **interprète** effectue lui même les opérations spécifiées par le programme source.

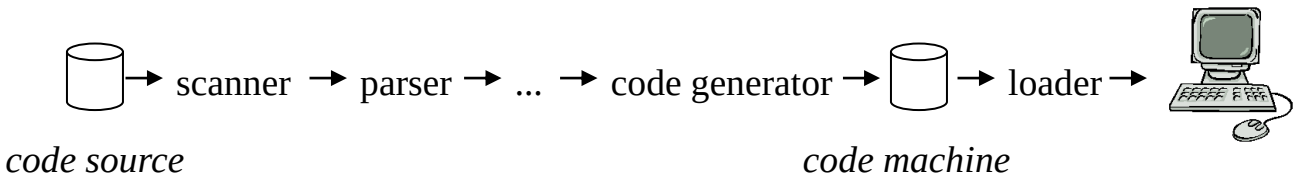
On utilise souvent des interprètes pour exécuter les langages de commande.

exemples:

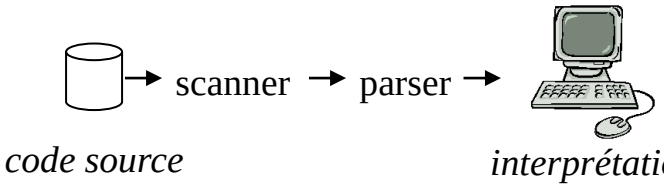
- Le *shell* d'Unix
- Le *command.com* du DOS

# Différence entre Compilateur et Interpréteur

**Compilateur** Traduit vers le code machine

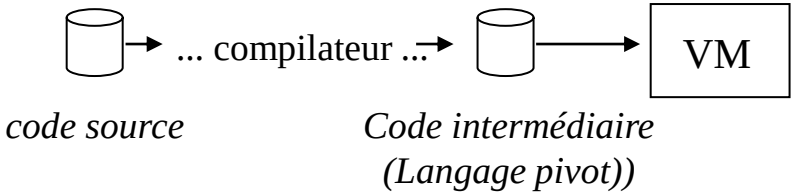



**Interpréteur** exécute le code source "directement"




- *Les instructions d'une boucle sont scannées et analysées à chaque itération*

**Variante**: interprétation du code intermédiaire




- *Le code source est traduit dans le code d'une machine virtuelle (VM)*
- *VM interprète le code simulant la machine physique*

# Exemple d'application

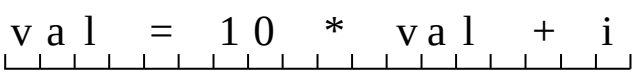
- Soit le programme source suivant:

Val = 10 * val + i

Schématisez les différentes phases de compilation.

# Exemple d'application

*Programme source*




Analyse lexicale

*Unités lexicales*

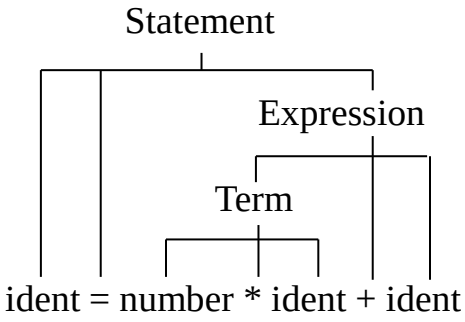
| 1 | 3 | 2 | 4 | 1 | 5 | 1 |
|---|---|---|---|---|---|---|
| (ident) "val" | (assign) - | (number) 10 | (times) - | (ident) "val" | (plus) - | (ident) "i" |

Code de l'unité

Valeur de l'unité

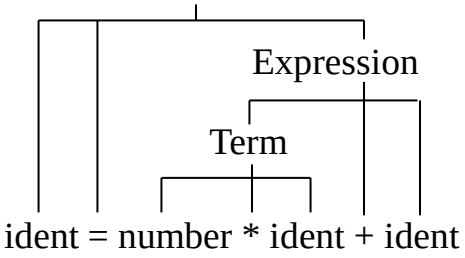
Analyse syntaxique

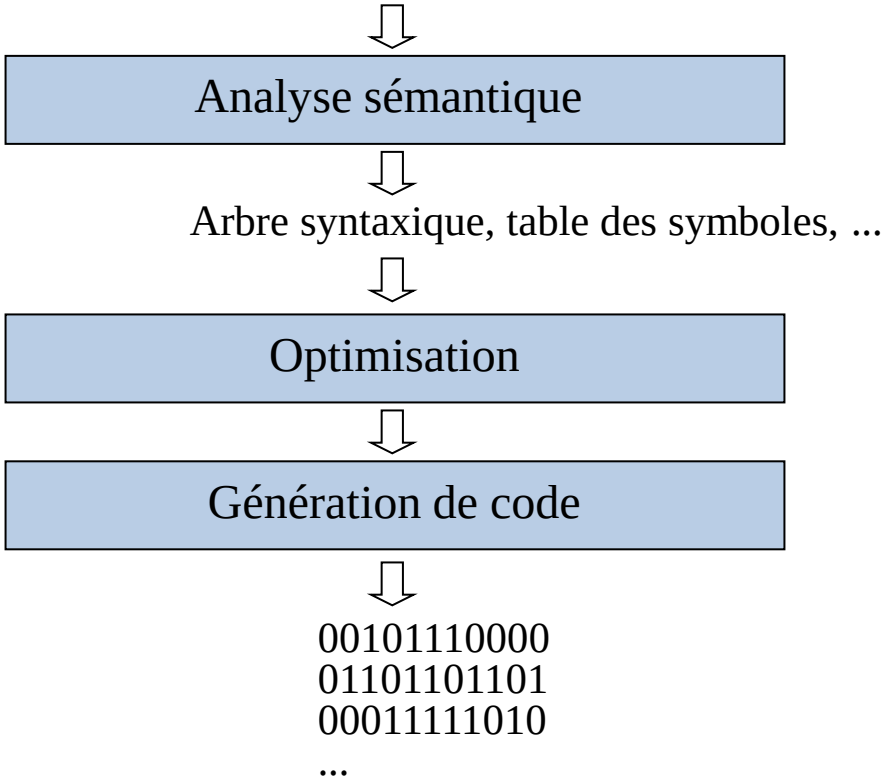
*Arbre syntaxique*



# Exemple d'application (suite)

*Arbre syntaxique*




*Représentation intermédiaire*

*Code machine*

00101110000
01101101101
00011111010
...

# The End